

**AFRL-IF-RS-TR-2002-297**  
**Final Technical Report**  
**November 2002**



# **REAL-TIME ASSET RESCHEDULING WITH EXECUTION MONITORING AND ACCURATE ASSET TRACKING**

**Kestrel Institute**

**Sponsored by**  
**Defense Advanced Research Projects Agency**  
**DARPA Order No. F080**

*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

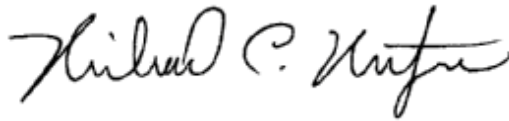
**The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.**

**AIR FORCE RESEARCH LABORATORY**  
**INFORMATION DIRECTORATE**  
**ROME RESEARCH SITE**  
**ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-297 has been reviewed and is approved for publication.

APPROVED:



RICHARD C. METZGER  
Project Engineer

FOR THE DIRECTOR:



MICHAEL L. TALBERT, Maj., USAF  
Technical Advisor, Information Technology Division  
Information Directorate

<b>REPORT DOCUMENTATION PAGE</b>			<i>Form Approved</i> <b>OMB No. 074-0188</b>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
<b>1. AGENCY USE ONLY (Leave blank)</b>		<b>2. REPORT DATE</b> NOVEMBER 2002	<b>3. REPORT TYPE AND DATES COVERED</b> Final May 97 – Sep 00	
<b>4. TITLE AND SUBTITLE</b> REAL-TIME ASSET RESCHEDULING WITH EXECUTION MONITORING AND ACCURATE ASSET TRACKING			<b>5. FUNDING NUMBERS</b> C - F30602-97-C-0154 PE - 63726F PR - F080 TA - T2 WU - 01	
<b>6. AUTHOR(S)</b> Douglas R. Smith, Stephen Fitzpatrick, and Stephen J. Westfold				
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Kestrel Institute 3260 Hillview Avenue Palo Alto California 94304			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> Defense Advanced Research Projects Agency AFRL/IFSE 3701 North Fairfax Drive 525 Brooks Road Arlington Virginia 22203-1714 Rome New York 13441-4505			<b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b>  AFRL-IF-RS-TR-2002-297	
<b>11. SUPPLEMENTARY NOTES</b>  AFRL Project Engineer: Richard C. Metzger/IFSE/(315) 330-7652/ Robert.Metzger@rl.af.mil				
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				<b>12b. DISTRIBUTION CODE</b>
<b>13. ABSTRACT (Maximum 200 Words)</b> Through the use of software transformational synthesis technology, synthesize rescheduling algorithms that reschedule transportation assets using data from automatic identification technology and asset tracking hardware and software. Kestrel brought to bear its software synthesis technology which allows for the generation of correct-by-construction, high performance schedulers from formal specifications of the problem they are intended to solve. Savi Technology brought to bear its expertise in automatic identification technology and asset tracking hardware and software systems. In this effort Kestrel and Savi co-developed scheduler for Yard Management Systems based on data from the Army's Crane Ammunition Depot.				
<b>14. SUBJECT TERMS</b> Software Transformational Synthesis, Scheduling Algorithms				<b>15. NUMBER OF PAGES</b> 50
				<b>16. PRICE CODE</b>
<b>17. SECURITY CLASSIFICATION OF REPORT</b>  UNCLASSIFIED	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b>  UNCLASSIFIED	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b>  UNCLASSIFIED	<b>20. LIMITATION OF ABSTRACT</b>  UL	

## Table of Contents

1 Project Goals and Chronology .....	1
2 Overview .....	2
3 Project Results .....	2
4 Real-time Tracking and Continuous Scheduling .....	5
5 CAMPS Mission Planning System .....	8
6 Planware .....	9
7 Concluding remarks .....	12
8 References .....	13
Appendix A Theories .....	15

## List of Figures

Figure 1: Continuous Scheduling Architecture .....	7
Figure 2: Taxonomy of Resource Theories .....	10
Figure 3: Spreadsheet-like Interface for Acquiring Scheduling Constraints .....	11

# 1 Project Goals and Chronology

This report describes our research on advanced scheduling technologies supported by the ARPA/Rome Lab Planning and Decision Aids (PDA) program during 1997 - 2000. At the outset, this project brought together Kestrel Institute and Savi Technology to develop technology for continuous scheduling based on highly-accurate real-time asset tracking and execution monitoring. The original goal was to demonstrate the benefits of combining high-quality logistics data with a robust scheduling capability in an operational DoD setting.

Kestrel brought to bear its software synthesis technology which allows the generation of correct-by-construction, high-performance schedulers from formal specifications of the problem they are intended to solve. Savi Technology, Inc. brought to bear its expertise in automatic identification technology and asset tracking hardware and software systems. Savi implemented numerous government and commercial asset visibility and tracking systems employing a wide range of automatic identification technologies (AIT), including bar-code, optical memory cards, smart cards, portable data terminals, and radio frequency identification. For example, Savi provides high-quality logistics data and execution monitoring for the U. S. Army and U. S. TRANSCOM in their Ammunition AIT project. We hoped to demo our combined results in this context.

Under this contract Kestrel and Savi codeveloped a continuous scheduler for Yard Management Systems, based on data from the Army's Crane Ammunition Depot. The Yard Management System project addresses the problem of scheduling a fleet of hostlers (tractors) in a dock-side depot. A typical depot has various types of temporary storage areas where cargo is held for inspection or until it is required for loading onto a ship, train or trailer for transportation out of the depot. The hostlers transfer cargo between the storage areas as the status of the cargo changes and between the storage areas and the cranes that transfer cargo onto or off ships and trains.

This demonstration system, while successful, didn't fully exploit the strengths of Kestrel's scheduling technology. One goal of this project was to piggyback on some of Savi's deployed systems to demonstrate the effectiveness of continuous scheduling in a context where Savi's realtime tracking technology had been deployed. As it happened, Savi's deployment schedule for its Ammo AIT project slipped, depriving the project of its main application domain. After consultation with all stakeholders, Kestrel and Rome Lab decided to terminate Savi's involvement and redirect Kestrel's research effort (in December 1998).

The revised project had two directions:

1. Support for the CAMPS project at AMC - Together with BBN and Carnegie-Mellon University, Kestrel developed an advanced scheduling system for AMC TACC (Air Mobility Command, Tanker-Airlift Operations Center) at Scott AFB. This system is scheduled for integration by Logicon Corporation into the CAMPS system and deployed in 2001.
2. Support for Planware - Planware is a domain-specific generator of high-performance scheduling software. The goal of the Planware system is to put Kestrel's scheduler

synthesis technology in the hands of users who aren't specialists in formal methods. Using Planware, the user interactively specifies a problem and then the system automatically generates a formal specification and refines it.

Funding problems disrupted this project in 1999. The Air Force terminated 6.3 funding service-wide in GFY 2000, leaving this project unfinished. Rome Lab was able to find some additional funds to continue Kestrel's CAMPS participation until May 2000. At that point, AMC was able to pick up support for CAMPS MPS.

## **2 Overview**

This report describes our research on the transformational development of scheduling software. Our approach to developing scheduling software involves several stages. The first step is to develop a formal model of the transportation scheduling domain, called a domain theory. Second, the constraints, objectives, and preferences of a particular scheduling problem are stated within a domain theory as a problem specification. Finally, an executable scheduler is produced semi-automatically by applying a sequence of transformations to the problem specification. The transformations embody programming knowledge about algorithms, data structures, program optimization techniques, etc. The result of the transformation process is executable code that is consistent with the given problem specification. Furthermore, the resulting code can be extremely efficient.

We developed this transformational technology and applied it to several application areas. First, we focused on continuous scheduling to sit on top of the real-time tracking systems supplied by Savi. Our previous schedulers had all been offline batch schedulers, so we were curious to extend our generative technology to cover continuous scheduling. We also wanted to exploit the availability of real-time data about the state of the world to continuously monitor and reschedule activities. The YMS demonstration system described in Section 4 embodies the results of this effort. Second, we used our transformation system KIDS [7, 9] to generate a scheduler for one of the world's most complex scheduling problems: strategic airlift scheduling at the Air Mobility Command at Scott AFB. This project is described in Section 5. Finally, we focused on incorporating our expertise in scheduler generation into a highly automated system for scheduler synthesis (Planware), which is described in Section 6.

## **3 Project Results**

This section provides a detailed list of significant accomplishments, events, demonstrations, and systems developed under this project. A compact summary of the most significant achievements of this project is given in the Concluding Remarks section.

1. YMS demonstration system - The Yard Management System project addresses the problem of scheduling a fleet of hostlers (tractors) in a dock-side depot. A typical depot has various types of temporary storage areas where cargo is held for inspection or until it is required for loading onto a ship, train or trailer for transportation out of

the depot. The hostlers transfer cargo between the storage areas as the status of the cargo changes and between the storage areas and the cranes that transfer cargo onto or off ships and trains. This system was demonstrated at the May 1998 and October 1998 PDA workshops.

2. *Planware* - A working version of Planware was developed and demonstrated extensively at workshops. Among the key developments are

(a) *Constraint Acquisition* - We developed a new tabular format for acquiring scheduling problem requirements. This result enabled a much richer means for interactively getting user input on the scheduling problem at hand, without requiring users to read or write formal specifications.

(b) *Colimits of Diagrams* - Another key technical result was the development and implementation of colimits of diagrams, which enabled us to provide precise support for composing structured representations of knowledge. We believe that this result is a major mathematical advance with respect to knowledge representation and processing, far transcending our particular application to program synthesis. We use colimits of diagrams to provide better support for design-by-classification, embodied in Planware's substrate called Designware (for more details, see [6, 5]). We applied for a patent on this operation.

(c) *Constraint Propagation Theory* - We developed and implemented a generalized theory of constraint propagation that subsumes earlier forms of constraint propagation (and-constraints) and global search (or-constraints).

3. *CAMPS Mission Planning System (MPS)* - We developed version 0.5 of MPS which was released at the end of the project. This version included basic scheduling of aircraft loading and routing, aircrews, and mog at enroute ports, as well as many other features required for deliberate airlift planning.

#### 4. *Publications*

Lee Blaine, Limei Gilham, Junbo Liu, Douglas R. Smith, and Stephen Westfold, Planware - Domain-Specific Synthesis of High-performance Schedulers, Proceedings of the Thirteenth Automated Software Engineering Conference, IEEE Computer Society Press, Los Alamitos, California, October, 1998, 270-280.

Smith, D.R., Mechanizing the Development of Software, in *Calculational System Design, Proceedings of the NATO Advanced Study Institute*, Eds. M. Broy and R. Steinbrueggen, IOS Press, Amsterdam, 1999, 251-292.

Smith, D.R., Designware: Software Development by Refinement, invited paper in *Proceedings of the Eighth International Conference on Category Theory and Computer Science*, Edinburgh, September, 1999.

Westfold, S.J. and Smith, D.R., Synthesis of Efficient Constraint Satisfaction Pro-

grams, Knowledge Engineering Reviews, Special Issue on AI and OR, 2001.

5. *Patents* - We applied for a patent on diagrams of colimits:

D. Pavlovic, J. Liu, and D.R. Smith, Method and Apparatus for Determining Colimits of Hereditary Diagrams, patent application submitted September 1999.

6. *Project-related Presentations, Meetings and Events* - Unless otherwise specified, Dr. Smith participated in each item below.

- CAMPS meeting, Scott AFB, 20 May 97.
- ARPI Annual Workshop, Lynnfield, MA, 9-12 June 1997; Dr. Smith received the *Guns and Butter Award* from the DARPA Program Manager for “basic research and a solid record of successful transitions to ARPI customers”. Dr Smith gave an invited talk entitled “Domain-Specific System for Synthesizing High-Performance Schedulers”; Dr. Smith gave demos of Planware and KIDS; Dr. Smith also gave a talk entitled “Synthesis of Real-Time Schedulers”.
- Interview with SAAM and Contingency & Exercise mission planners, AMC HQ, Scott AFB, 29 September 97.
- CAMPS meeting, Logicon HQ, San Pedro, CA, 27-28 October 1997.
- PDA Workshop, San Francisco, 4-6 Nov 97.
- CAMPS meeting, interview with SAAM and Contingency & Exercise mission planners, AMC HQ, Scott AFB, 3 December 97.
- visited MTMC HQ, Falls, Church, VA to discuss KI/Savi scheduling of yard management at Sunny Point. Attending: Col Larry Curtin, Col Morrow, Dan Monahan, 16 January 1998.
- Talk and Demo of CAMPS Mission Planner, Logicon Office, Scott AFB, 28 January 1998.
- Visited Sunny Point, NC to interview MOTSU personnel, 17-18 Feb 98. Result was hostility to Kestrel/Savi participation in an application effort, so we looked elsewhere.
- CAMPS meeting, Kestrel, 4 Mar 98
- CAMPS meeting, Logicon HQ, San Pedro, 5 Mar 98
- CAMPS demos and discussions, Logicon at Scott AFB, 28-29 Apr 98.
- DARPA ISO meeting, Monterey, CA, 4-5 May 98. Presentation of the Yard Management System, CAMPS, and Planware at ARPI Annual meeting, Monterey, CA, 6-7 May 1998.
- CAMPS meeting, Logicon HQ, San Pedro, 19-20 May 98 (Tom Emerson attended)
- PDA program review at Kestrel with Al Frantz, 25 Jun 98.
- Talks on Planware and category theory applied to software refinement, Air Force Rome Lab, Rome, NY, 2 Sept 98.
- Tutorial on Designware and paper on Planware, Automated Software Engineering, 13-16 Oct 98, Honolulu, HI.
- Talks and demos on Planware, CAMPS, and Continuous Scheduling Architecture, ARPI meeting, Arlington VA, 27-29 Oct 98.
- CAMPS meeting, AMC, Scott AFB, 17-19 Feb 99.



- Project review, PDA, John Lemmer, 30 Apr 99
- Meeting on USAF Common Scheduler Project, ESC, Hanscom AFB, Bedford, MA, 21 May 99.
- Invited Talk, "Software Development by Mechanized Refinement", Workshop on Language, Logic, and Computation, CSLI, Stanford University, 29 May 1999.
- Meeting on ESC Common Scheduler Project, at Kestrel, 12 Aug 99. Attending were Frank LaMonica and John Lemmer (AFRL), Russ Graves (MITRE & ESC), and Kestrel personnel.
- Invited Talk, Designware, Eighth International Conference on Category Theory and Computer Science (CTCS 99), Univ. Edinburgh, UK, 10-12 September 1999.
- talk on Planware, and CAMPS demo, ARPI/PDA Workshop, Arlington VA, 14-15 September 1999.
- Invited Talk, CS Colloquium, Cornell University, 30 Sept 99.
- Invited talk on Designware, High Integrity Systems Conference, Albuquerque, NM, 14-17 Nov 99.
- Invited talk on Planware, to Air Force Scientific Advisory Board, AFRL, Rome, NY, 7-8 Dec 99.
- CAMPS meeting, Kestrel, 13-14 April 2000.
- CAMPS meeting, Logicon Scott office, O'Fallon, IL, 11-12 May 2000.
- BBN, Cambridge, MA, CAMPS work, 2 Jun 00
- CAMPS RAD/JAD, alpha release of CAMPS MPS, Logicon Scott, 27-28 Jun 00

## 4 Real-time Tracking and Continuous Scheduling

The Yard Management System project addresses the problem of scheduling a fleet of hostlers (tractors) in a dock-side depot. A typical depot has various types of temporary storage areas where cargo is held for inspection or until it is required for loading onto a ship, train or trailer for transportation out of the depot. The hostlers transfer cargo between the storage areas as the status of the cargo changes and between the storage areas and the cranes that transfer cargo onto or off ships and trains.

The objective in scheduling the fleet of hostlers is to reduce the loading and unloading times of ships and potentially to achieve acceptable loading and unloading times with fewer hostlers. The main difficulty in constructing an optimal schedule is that the number of schedules for any given set of cargo movements is exponential in the number of movements: for even moderately large numbers of movements, it is infeasible to try every possible schedule.

One solution is to compute lower bounds on the completion time for schedules and use this information to prune from the schedule search space schedules that could not possibly be optimal. The lower bound computations typically require much less time than would the construction of the schedules themselves. Furthermore, a given lower bound computation typically applies to a large number of schedules (derived from a common, partially constructed

schedule) all of which may be pruned. Lower bound computations can thus make the construction of an optimal schedule feasible.

### *The YMS system*

We developed a prototype YMS system that provides continuous scheduling of logistics operations by means of near-real time asset visibility data, sentinels and monitoring code, continuous scheduling engine, and an execution manager (see Figure 1).

We describe the various components of this architecture below. The components are processes running concurrently on a Unix workstation, and communicating via CORBA middleware infrastructure.

### *Savi Asset Manager*

The Savi Asset Manager provides near-real time asset data acquisition based on automatic identification technologies. The asset data is gathered in various distributed and central locations.

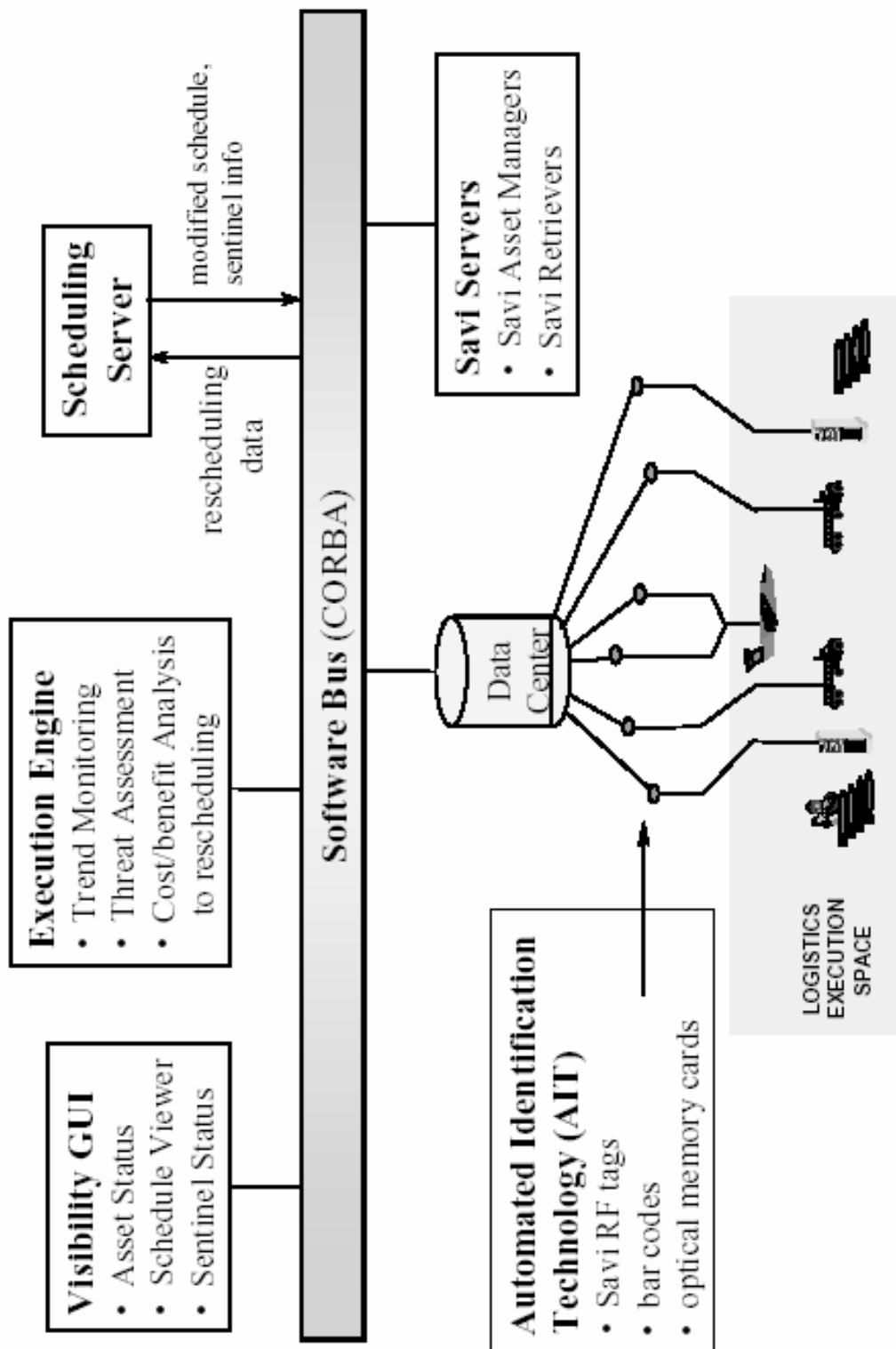
### *Scheduling Server*

The scheduling engine transforms the available resources, delivery requirements, and domain constraints into a schedule for logistics operations. Several rescheduling algorithms are available - global reschedulers for times when it is more important to reoptimize the schedule than to minimize change, and local reschedulers when minimizing change is desirable. We anticipate supporting user-specifiable degrees of optimization versus change in the scheduling server. The scheduling engine has the ability to determine sensitive events or combination of events that could require rescheduling operations. Under the control of user-specified parameters and other forms of guidance, the scheduling server automatically generates sentinels to watch for particular events.

### *Execution Manager*

The execution manager provides situation visibility and control to the user, and mediates the activities of the data tracking system and the scheduling server.

Visibility and control: The status of the assets is provided via direct access to the data center. The Schedule viewer provides a view of the scheduled logistics operations, for a record of recent changes to the logistics operations, and information about the capabilities, delivery requirements, and constraints. The Sentinel Status view provides information of currently active sentinels and previously triggered sentinels. The user specifies which sentinels to generate and how. The user can also choose from a range of rescheduling tools and tool guidance following notification of a need to reschedule from a sentinel.



*Figure 1: Continuous Scheduling Architecture*

Mediating the tracking system and scheduler: Sentinels are executed by an interpreter in the execution engine that creates a process that watches (active sentinel) for a given set of events. The sentinel remains active until the events happen or the sentinel is canceled or replaced. If a sentinel event occurs, the execution manager notifies the user and may trigger the replanning operation.

Kestrel researchers constructed a domain theory for the YMS scheduler that models the typical operation of a depot and containing algebraic laws that permit the automated derivation of lower bound functions suited to the depot (See Appendix A). Furthermore, the domain theory contains laws that permit the automated instantiation of various algorithm theories such as global search, local search and greedy heuristic algorithms. Instantiation of these theories would produce a family of schedulers having various degrees of compromise between scheduler execution time and the quality of the schedules produced (not all of the schedulers will produce optimal schedules). The family of schedulers would also contain real-time reschedulers which use information provided by real time tracking systems to adjust schedules as they are executed to account for deviations between predicted and actual performance.

Successful demonstrations of the YMS system were given at the ARPI workshops in May 1998 and October 1998.

## 5 CAMPS Mission Planning System

This section provides a brief overview of the algorithm underlying the Mission Planner in CAMPS. The MPS algorithm operates on level 2 TPFDD data and generates mission plans over nominal aircraft and crews. The MPS tracks and constrains the following kinds of resources in constructing a mission plan: (1) aircraft, (2) crews and their duty days, (3) parking mog, (4) working mog, (5) runways (takeoff and landing separations), (6) port operating hours. Currently, fuel supplies at ports are tracked but not constrained.

The MPS algorithm essentially iterates over movement requirements which are sorted by increasing LAD primarily.

1. *The main control loop:*

For each movement requirement (MVR), the algorithm attempts to move it in the context of previously planned missions. First, the algorithm tries to place the MVR on existing mission provided that (1) it has capacity, (2) the mission route includes the POE and POD, (3) the mission is feasible with respect to the ALD, EAD and LAD of the MVR. If an existing mission cannot (fully) carry the MVR, then the algorithm will attempt to create a new mission.

2. *Creating a new mission:*

Given MVR, we first look ahead in the list of unscheduled movement requirements to find those that could be coalesced with it; in particular, we look for requirements from the same (or a nearby) POE with a similar ALD that are going to the same POD. If

enough cargo (or PAX) can be found to justify a new mission (according to a minimum threshold per MDS), then a route generator is called to produce a candidate list of available aircraft and routes. Route generation includes positioning from the current location of the aircraft to the POE, travel through enroutes (including possible AR stops) to the POD, followed optionally by deposition to a recovery base. For each leg of the mission, the algorithm generates choices of (1) crew and whether the crew flies the leg in their current duty or the next, (2) base open hours interval for departure, (3) takeoff position (to enforce takeoff separations), (3) base open hours interval for arrival, and (4) landing position (to enforce landing separations). If mog bounds are specified for ports along the route, then parking mog and working mog are tracked and the constraints are enforced.

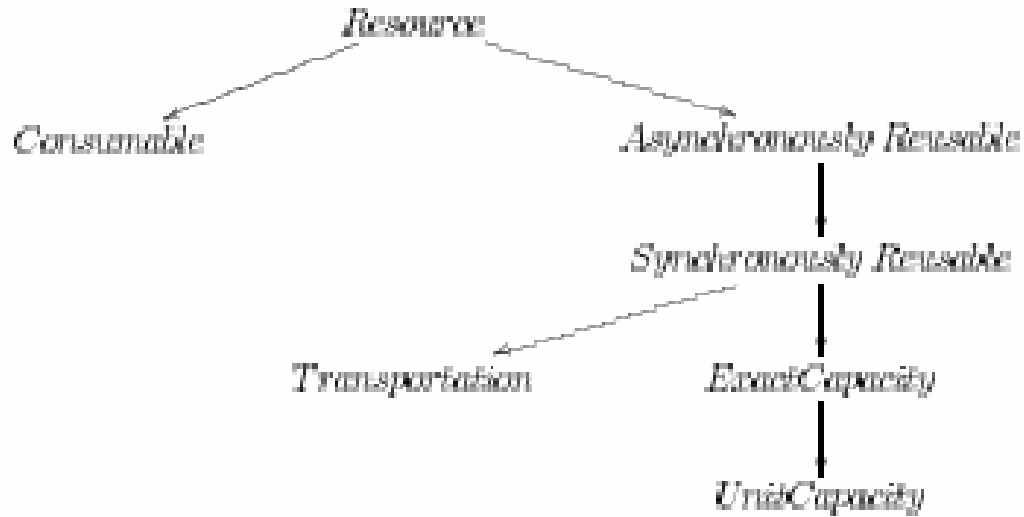
The Mission Planner is based on constraint propagation technology. See [9] for more details about the nature and design of constraint-based algorithms for scheduling. One key feature of our approach is to use time windows for every temporal decision; in particular during scheduling, the departure time of every flight is given by a time window - an earliest departure time and a latest departure time. When the departure times of flights must be coordinated, for example, by enforcement of a minimum separation between takeoffs at a port, then a dependence constraint is asserted between flights so that the earliest departure time of one flight is, say, at least 15 minutes before the earliest departure time of another flight departing from the same station. If the departure time of the first flight must be delayed, then the asserted constraint will cause the earliest departure time of the second flight to be delayed if necessary to enforce the minimum takeoff separation. In this scenario a change to the time window of one flight is propagated through the takeoff separation constraint to cause a change to the time window of the second flight - an example of constraint propagation.

## 6 Planware

This section presents an overview of Planware, a generator of high-performance scheduling algorithms, currently being developed at Kestrel Institute. Our aim is to convey a sense of the rationale for Planware, the design process that it supports, the architecture of the current Planware system, and our results to date. The reader may find more detail in the references.

Architecturally, Planware is an extension of the Specware system [10], a system for developing formal specifications and refinements based on concepts from higher-order logic and category theory. Planware and Specware embody theoretical developments stemming from Kestrel's experience with previous systems, such as KIDS [7] and DTRE [2].

The goal of Planware is to allow experts in planning and scheduling to assemble quickly a specification of a scheduling problem, and to generate automatically a high-performance scheduler from it. The user's interactions with the system are designed to be entirely in the scheduling domain { the user does not need to read or write formal specifications, nor to understand the logical and category-theoretic foundations of the system. We have invested substantial effort in automating the construction of scheduling domain theories.



**Figure 2: Taxonomy of Resource Theories**

To assemble a requirement specification and underlying domain theory, Planware requires very little information from the user:

1. to select from a taxonomy of resource theories the particular kind of resource against which to schedule the tasks (See Figure 2)
2. to modify a spreadsheet-like table of expressions that provide bounds on the various attributes of reservations of the selected resources (See Figure 3).

From this minimal amount of information, Planware can automatically

- generate a formal specification of the scheduling problem (plus the relevant background concepts that comprise a domain theory),
- reformulate the specification using datatype refinements to build some of the problem constraints directly into the schedule datatype, allowing a dramatic simplification of the specification,
- apply domain-independent knowledge about designing global search (backtracking) algorithms with constraint propagation,
- apply datatype refinements and optimization techniques, and finally
- generate Common Lisp code.

<b>Parameter</b>	<b>Lower Bound</b>	<b>Exact Value</b>	<b>Upper Bound</b>
<i>Start Time</i>	<i>Task.release</i>	<i>Finish - Dur</i>	<i>Task.pick-up</i>
<i>Resource-type</i>		<i>{C130, C141, C5, C17,..}</i>	<i>Sum of task req'd resources</i>
<i>Instantaneous Demand</i>	<i>min-cap</i>	<i>Sum of task demands</i>	<i>max-cap</i>
<i>Duration</i>	<i>0</i>	<i>Finish - Start, Dist(orig,dest)/speed</i>	
<i>Finish Time</i>	<i>Task.ead</i>	<i>Start + Dur</i>	<i>Task.due-date</i>
<i>Max-capacity</i>		<i>r.r-type.max-cap</i>	
<i>Separation</i>	<i>0</i>	<i>r.r-type.on-ground-time, r.r-type.enroute-stop</i>	
<i>Origin</i>		<i>Task.poe</i>	
<i>Speed</i>		<i>r.type.blockspeed</i>	

Also: *Precedes, Min-capacity, Destination*

**Figure 3: Spreadsheet-like Interface for Acquiring Scheduling Constraints**

For example, after design and refinement, the specification of a transportation scheduling domain comprises about 10,000 lines of text of which about 3000 lines are the scheduling algorithm (the remainder consists of axioms and datatype operations that are not needed by the running scheduler).

A key point here is that the high level of automation in Planware is achieved by applying domain- specific control (via a hand-built tactic):

1. to construct a problem specification and domain theory,
2. to apply a series of domain-independent design theories and code-generation rules. The result is a fast, correct, executable scheduler automatically constructed from the user's description of a scheduling problem.

We believe that Planware is a new paradigm for domain-specific software generators. Planware differs from other domain-specific software generators in that it is built on a foundation of domain-independent general-purpose software specification and synthesis capabilities

(Specware/Designware). In particular, Planware relies on

1. the Specware capabilities for composing specifications, refining them and translating to code (especially colimits of diagrams);
2. the Designware libraries of domain-independent design knowledge about algorithms, datatype refinements, and expression optimization techniques (and their application tactics) to construct refinements.

The domain-specificity of Planware comes in the form of

**specifications of domain knowledge** - the abstract scheduling specification, the taxonomies of task and resource theories, etc.

**scheduling spec construction tactics** - tactics for lifting properties of tasks to constraints on the scheduler, tactics for lifting resource constraints to scheduling constraints, tactics for constructing the constructors and other datatype operations needed by the refined Task, Resource, Reservation, and Schedule specs,

**embeded algorithm design tactic** - tactics for generating a global search theory for the scheduling problem at hand, etc.

The background of domain-independent design knowledge allows a user to derive software even when the requirements fall outside the domain-specific scope of the system. The user then gets less automation, and must supply more guidance in the construction process.

There are many things to be done before Planware can be deployed. One crucial extension is allowing the user more flexibility in supplying task information. We are developing a spreadsheet-like interface that is derived from the user's choice of resource theory and presents the user with plausible options for lower/exact/upper bounds on all crucial attributes of a reservation for that kind of resource. We are working to let the user choose and modify arbitrary entries. As before the user only interacts with the system in domain-specific terms. Another vital extension is to generalize the abstract scheduling spec to multiple resource classes. Another extension that is underway is to extend Planware to allow the synthesis of scheduling systems, including visual displays, editors, GUI, database mediators, and so on.

See [1] for more details and an example of Planware in action.

## 7 Concluding remarks

This project enabled us to show the flexibility of our specs-to-code technology as applied to a range of scheduling problem types: continuous scheduling, large-scale complex airlift scheduling, and highly-automatic scheduler generators. We expect each of these areas to continue to develop and to result in improved products for DoD needs.



Summing up, this project has led to theoretical breakthroughs, direct application support for the US Air Force, and business opportunities in technology transfer:

*1. Fundamental Breakthroughs in Software Synthesis Technology -*

- *Diagram Colimit* - This is a fundamental tool for automatically composing knowledge structures. Our belief in the long-term criticality of this operation is reflected in our applying for a patent on it.
- *Planware* - Planware is a new paradigm for domain-specific software generators. Planware differs from other domain-specific software generators in that it is built on a foundation of domain-independent general-purpose software specification and synthesis capabilities (e.g. the diagram colimit operation in Specware/Designware). Our discovery of the tabular spreadsheet-like format for acquiring problem requirements was a significant breakthrough in the Planware interface. It is paradigmatic of how a simple domain-specific specification interface can be built on the basis of a clear mathematical model of a class of problems.

*2. Application Support for the US Air Force* - CAMPS MPS is the next-generation scheduling system at the TACC at AMC, Scott AFB. It provides speed and scheduling capabilities far beyond AMC's current tools. MPS' support for hub-and-spoke operations may enable a radical change in AMC airlift operations strategy.

*3. Tech Transfer and Business Opportunities* - Planware as a generator (vending machine) of schedulers has generated commercial interest. Kestrel Technology (a for-profit LLC started recently to develop and transfer Kestrel Institute technologies) has won an SBIR to develop Planware into a commercially viable tool. We also anticipate that Kestrel Technology will provide long-term maintenance support to AMC for CAMPS MPS, and will be looking for other opportunities for developing variants of MPS for the US Air Force.

## 8 References

- [1] Blaine, L., Gilham, L., Liu, J., Smith, D. R., and Westfold, S. Planware { domain-specific synthesis of high-performance schedulers. In Proceedings of the Thirteenth Automated Software Engineering Conference (October 1998), IEEE Computer Society Press, pp. 270 - 280.
- [2] Blaine, L., and Goldberg, A. DTRE - a semi-automatic transformation system. In Constructing Programs from Specifications, B. Möller, Ed. North-Holland, Amsterdam, 1991, pp. 165 - 204.
- [3] Meseguer, J. General logics. In Logic Colloquium 87, H. Ebbinghaus, Ed. North Holland, Amsterdam, 1989, pp. 275 - 329.
- [4] Smith, D. R. Toward a classification approach to design. In Proceedings of the Fifth

International Conference on Algebraic Methodology and Software Technology, AMAST'96 (1996), vol. LNCS 1101, Springer-Verlag, pp. 62 -84.

- [5] Smith, D. R. Designware: Software development by refinement. In Proceedings of the Eighth International Conference on Category Theory and Computer Science (1999), M. Hoffman, D. Pavlovic, and P. Rosolini, Eds., pp. 355-370.
- [6] Smith, D. R. Mechanizing the development of software. In Calculational System Design, Proceedings of the NATO Advanced Study Institute, M. Broy and R. Steinbrueggen, Eds. IOS Press, Amsterdam, 1999, pp. 251 - 292.
- [7] Smith, D. R. KIDS - a semi-automatic program development system. IEEE Transactions on Software Engineering Special Issue on Formal Methods in Software Engineering 16, 9 (September 1990), 1024-1043.
- [8] Smith, D. R., and Lowry, M. R. Algorithm theories and design tactics. In Proceedings of the International Conference on Mathematics of Program Construction, LNCS 375, L. van de Snepscheut, Ed. Springer-Verlag, Berlin, 1989, pp. 379-398. (reprinted in Science of Computer Programming, 14(2-3), October 1990, pp. 305-321).
- [9] Smith, D. R., Parra, E. A., and Westfold, S. J. Synthesis of planning and scheduling software. In Advanced Planning Technology (1996), A. Tate, Ed., AAAI Press, Menlo Park, pp. 226-234.
- [10] Srinivas, Y. V., and Jüllig, R. Specware: Formal support for composing software. In Proceedings of the Conference on Mathematics of Program Construction, B. Moeller, Ed. LNCS 947, Springer-Verlag, Berlin, 1995, pp. 399-422.

# Appendix A Theories

## A YMS Theories

This appendix details the KIDS theories for the Yard Management System scheduler. A brief summary of the theories is presented below, followed by the full text of the theories.

**Cargo** A simple ADT representing a cargo item.

**Slot** The various types of storage area in a yard (bunkers, loading bays, etc.) are uniformly represented as 'slots'.

**Movement Request** This ADT represents a task for the hostlers in the form of an item of cargo to be moved from a source slot to a destination slot. Some additional information, such as the distance between source and destination fields is cached for efficiency in computing metrics on schedules.

**Ship Manifest** This represents a partial order on movement requests that arises from physical restrictions on loading and unloading ships. The movement requests are categorized into numbered strata such that all movement requests in a given strata must be accomplished before any movement requests from higher-numbered strata, but there is no ordering on the movement requests within a single strata.

**Geometry** The geometry of a yard is represented essentially as a graph of distances between slots. The graph is directed to allow for asymmetric routes (e.g., one-way roads).

**Hostler** The Hostler theory defines an ADT to represent individual hostlers and their properties; e.g., their speed and turn-around times.

**Yard** A Yard represents a single physical scheduling site: its geometry (slots and the distances between them) and resources (hostlers).

**Reservation** A reservation represents the assignment of a hostler to satisfy a particular movement request during a particular time period.

**Schedule** A schedule represents the assignments of all hostlers over a certain time period. Conceptually, a schedule is a set of reservations.

**Work Metric** The work metric is used to assess schedules both in gauging the quality of a schedule and in computing lower bounds on completions of partial schedules (these lower bound are used for pruning searches). The work represented in a schedule is based on productive work (cargo moved) and unproductive work (repositioning of hostlers between cargo movements).

**YMS Theory** This theory defines feasibility on hostler schedules. The primary components of feasibility are: the ordering on cargo movements is respected; each reservation is long enough for its cargo movement to be accomplished; successive pairs of reservations for a given hostler are sufficiently separated to allow for repositioning of the hostler; the schedule does not allocate more hostlers than are available in the yard.

**Global Search with Pruning** This theory defines an algorithm schema for performing global search with pruning based on a lower bound on the costs of all schedules in a given branch of the search space. The schema includes a parameter that allows the quality of the final schedule to be specified in terms of deviation from true optimality (e.g., 'within 5% of optimal'). Reducing the quality even slightly from strict optimality can dramatically reduce the size of the search space. The theory also shows how the search can be initialized with a schedule generated by a quick but non-optimizing heuristic scheduler to further reduce the size of the search space.

**Greedy Append for Earliest Finish** This theory is an example of a greedy heuristic scheduler that can be used to initialize the pruning global search scheduler. It is based on constructing reservations to minimize the finishing time for some movement request chosen at random from the head of the manifest (such that the ordering of the movement requests is respected).

## A.1 Cargo

```
!! in-package("RE")
!! in-grammar("THEORY-GRAMMAR, 'cypress-grammar")
```

% Represents an item of cargo.

THEORY CARGO

% -----

THEORY-IMPORTS {}

% -----

THEORY-TYPE-PARAMETERS {}

% -----

THEORY-TYPES

```
type cargo-id = symbol
type cargo-type = symbol
```

```

type cargo = tuple(id: cargo-id, cargo-type: cargo-type)

% -----

THEORY-OPERATIONS
function cg-make(id: cargo-id, cargo-type: cargo-type): cargo
  = <id, cargo-type>

function cg-id(c: cargo): cargo-id
  = c.id

function cg-type(c: cargo): cargo-type
  = c.cargo-type

END-THEORY

```

## A.2 Slot

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Represents a "slot" (hostler berth) in a yard.
THEORY SLOT

% -----

THEORY-TYPES

type slot-id = fixnum
type slot-name = symbol
type slot = tuple(id: slot-id, name: slot-name)

% -----

THEORY-OPERATIONS
function sl-make(id: slot-id, name: slot-name): slot
  = <id, name>

function sl-id(s: slot): slot-id
  = s.id

function sl-name(s: slot): slot-name
  = s.name

```

END-THEORY

### A.3 Movement Request

```
!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)
```

% Represents a request to move cargo.

THEORY MOVEMENT-REQUEST

% -----

THEORY-IMPORTS {SLOT, CARGO, GEOMETRY}

% -----

THEORY-TYPES

```
type mvr = tuple(source: slot-id,
                 destination: slot-id,
```

```
cargo: cargo,
tag: fixnum,
dist: distance,
load: fixnum)
```

% -----

THEORY-OPERATIONS

```
function mvr-make(s: slot-id, d: slot-id, c: cargo): mvr
= <s, d, c, undefined, undefined, undefined>
```

```
function mvr-source(m: mvr): slot-id
= m.source
```

```
function mvr-destination(m: mvr): slot-id
= m.destination
```

```
function mvr-cargo(m: mvr): cargo
= m.cargo
```

```

function mvr-set-tag(m: mvr, tag: fixnum): mvr
  = <mvr-source(m),
    mvr-destination(m),
    mvr-cargo(m),
    tag,
    m.dist,
    m.load >

function mvr-tag(m: mvr): fixnum
  = m.tag

function mvr-compute-distance(m: mvr, g: geo): mvr
  = let(s: slot-id = mvr-source(m),
d: slot-id = mvr-destination(m))
    let(dist: distance = geo-distance(g, s, d))
    <s, d, mvr-cargo(m), mvr-tag(m),
    dist,
    (dist div *LOADED-HOSTLER-SPEED* + *TRIP-DURATION-OVERHEAD*) >

function mvr-distance(m: mvr, g: geo): distance
  = if defined?(m.dist)
    then m.dist
    else geo-distance(g, mvr-source(m), mvr-destination(m))

function mvr-cached-distance(m: mvr): distance
  = m.dist

function mvr-load(m: mvr): fixnum
  = m.load

```

% -----

## THEORY-LAWS

```

assert mvr-source-and-destination-different
  fa(mvr) mvr-source(mvr) ~ = mvr-destination(mvr)

```

END-THEORY

## A.4 Ship Manifest

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

```

## THEORY SHIP-MANIFEST

% -----

THEORY-IMPORTS {TIME-AND-DISTANCE, MOVEMENT-REQUEST}

% -----

THEORY-TYPE-PARAMETERS {}

% -----

## THEORY-TYPES

```
type rank = fixnum
type repo-map = tuple(index: seq(fixnum),
                      distances: seq(distance))
type manifest = tuple(rank: fixnum,
                      loaded: set(mvr),
                      unloaded: set(mvr),
                      unloaded-distance: distance,
                      repo: repo-map
                      )
```

% -----

## THEORY-OPERATIONS

```
function man-make-empty(): manifest
= <0, {}, {}, 0, undefined>
```

```
function man-rank(man: manifest): fixnum
= man.rank
```

```
function man-add-stratum(man: manifest, mvrs: set(mvr)): manifest
= let(rank = man-rank(man)+1)
  <rank,
    man.loaded,
    (man.unloaded union
      image(lambda(m: mvr) mvr-set-tag(m, rank), mvrs)),
    undefined,
    undefined>
```

```
function man-loaded(man: manifest): set(mvr)
= man.loaded
```



```

function man-unloaded(man: manifest): set(mvr)
  = man.unloaded

function man-finished?(man: manifest): boolean
  = man-unloaded(man) = {}

function man-rank-of-mvr(man: manifest, m: mvr): fixnum
  = mvr-tag(m)

function man-mvr-less?(man: manifest, mvr1: mvr, mvr2: mvr): boolean
  = mvr-tag(mvr1) < mvr-tag(mvr2)

function man-mvr-has-been-loaded?(man: manifest, mvr: mvr): boolean
  = mvr in man-loaded(man)

function man-remove-mvr-from-distance-map(man: manifest, done: mvr)
  : repo-map
  = if not(defined?(man.repo))
    then undefined
    else <man.repo.index,
        seq-shadow1(man.repo.distances, mvr-tag(done), 0)>

function man-mark-mvr-loaded(
  man: manifest, mvr: mvr | mvr in man-unloaded(man)
): manifest
  = (<man.rank,
    (man.loaded with! mvr),
    (man.unloaded less! mvr),
    (man.unloaded-distance - mvr-load(mvr)),
    man-remove-mvr-from-distance-map(man, mvr)>)

function man-head(man: manifest): set(mvr)
  = let(min-rank: fixnum
    = min-i(image(lambda(m:mvr) man-rank-of-mvr(man, m),
man-unloaded(man))))
    filter(lambda(m: mvr) man-rank-of-mvr(man, m) = min-rank,
man-unloaded(man))

function man-tail(man: manifest): set(mvr)
  = let(min-rank: fixnum
    = min-i(image(lambda(m:mvr) man-rank-of-mvr(man, m),
man-unloaded(man))))
    filter(lambda(m: mvr) man-rank-of-mvr(man, m) > min-rank,
man-unloaded(man))

```

```

function man-refined?(man1: manifest, man2: manifest): boolean

function man-compute-unloaded-distance(man: manifest, g: geo)
  : manifest
= let(new-unloaded: set(mvr) =
image(lambda(m: mvr) mvr-compute-distance(m, g), man-unloaded(man)))
  let(dist: distance =
reduce(+, image(mvr-load, set-to-seq(new-unloaded))))
  <man.rank, man.loaded, new-unloaded, dist, man.repo>

function man-unloaded-distance(man: manifest): distance
= man.unloaded-distance

function man-mvrs-in-order(man: manifest): seq(mvr)
= if man-finished?(man)
  then []
  else
    let(m: mvr = arb(man-head(man)))
    prepend(man-mvrs-in-order(man-mark-mvr-loaded(man, m)), m)

function man-find-closest-mvr(m: mvr, s: seq(mvr), i: fixnum, g: geo)
  : tuple(mvr, distance, mvr)
= if i=size(s)
  then <m, 0, undefined>
  elseif i=size(s) - 1
  then <m,
    geo-distance(g, mvr-destination(m), mvr-source(last(s))),
    last(s)>
  else
    let(md: tuple(mvr, distance, mvr)
= reduce(lambda(x: tuple(mvr, distance, mvr),
y: tuple(mvr, distance, mvr))
  if x.2 <= y.2 then x else y,
image(lambda(j)
  <m,
  geo-distance(g, mvr-destination(m), mvr-source(s(j))),
  s(j)>,
[i + 1 .. size(s)])))
md

function man-find-closest-mvrs-aux(todo: seq(mvr), g: geo,
done: seq(mvr), chosen: map(mvr, set(mvr)),
closests: seq(tuple(mvr, distance, mvr))
): seq(tuple(mvr, distance, mvr))
= if empty(todo)
  then closests

```

```

else
  let(next: mvr = first(todo))
  let(candidates: seq(tuple(distance, mvr))
    = [ <d,m> | (m: mvr, d: distance)
m in done
& (m in domain(chosen)
  => fa(c: mvr)(c in chosen(m)
=> mvr-destination(c) ~ = mvr-destination(next)))
& d = geo-distance(g, mvr-destination(next), mvr-source(m)) ])
  let(dc: tuple(distance, mvr)
    = reduce(lambda(dm1: tuple(distance, mvr),
                    dm2: tuple(distance, mvr))
              if dm1.1 <= dm2.1 then dm1 else dm2,
              candidates))
  let(choice: mvr = dc.2,
dist: distance = dc.1)
  man-find-closest-mvrs-aux(
rest(todo),
g,
append(done, next),
chosen
  + * { | choice -> if defined?(chosen(choice))
                    then chosen(choice) with next
                    else {next} | },
prepend(closests, <next, dist, choice> ))

function man-find-closest-mvrs(mvrs: seq(mvr), g: geo)
: seq(tuple(mvr, distance, mvr))
= let(rev: seq(mvr) = reverse(mvrs))
  let(m: mvr = first(rev))
  let(e = <m, 0, undefined>)
  man-find-closest-mvrs-aux(rest(rev), g, [m], { | | }, [e])

function man-compute-closest-sources(man: manifest, g: geo): manifest
= let(mvrs: seq(mvr) = man-mvrs-in-order(man))
  let(closest: seq(tuple(mvr, distance, mvr))
    = man-find-closest-mvrs(mvrs, g))
  let(sorted: seq(tuple(mvr, distance, mvr))
    = sort(copy-list(closest),
lambda(x: tuple(mvr, distance, mvr), y: tuple(mvr, distance, mvr))
  x.2 >= y.2))
  let(index: seq(fixnum)
    = image(lambda(x: tuple(mvr, distance, mvr)) mvr-tag(x.1), sorted))
  let(distances: seq(distance)
    = image(lambda(x: tuple(mvr, distance, mvr)) x.2, closest))
  (< man.rank, man.loaded, man.unloaded, man.unloaded-distance,

```

<index, distances > >)

function smaller-i(x: fixnum, y: fixnum): fixnum  
= if x < y then x else y

function sum-n-non-zeros(s: seq(distance), index: seq(fixnum),  
n: fixnum, pos: fixnum, total: distance  
): distance  
= if pos > size(index) or n=0  
then total  
elseif s(index(pos)) = 0  
then sum-n-non-zeros(s, index, n, pos+1, total)  
else sum-n-non-zeros(s, index, n-1, pos+1, total+s(index(pos)))

function man-repositioning-lower-bound(man: manifest, Y: yard)  
: distance  
= reduce(+, man.repo.distances, 0)  
- sum-n-non-zeros(man.repo.distances, man.repo.index,  
smaller-i(size(y-hostler-ids(Y)) - 1,  
size(man.repo.index)),  
1, 0)

% -----

## THEORY-LAWS

assert man-head-and-tail-disjoint  
fa(m) man-head(m) intersect man-tail(m) = {}

assert man-head-and-tail-cover-unloaded  
fa(m) man-head(m) union man-tail(m) = man-unloaded(m)

assert man-equivalence-of-loaded-and-mvr-has-been-loaded  
fa(m) man-loaded(m)  
= { mvr | (mvr) man-mvr-has-been-loaded?(m, mvr) }

assert man-loaded-and-unloaded-are-disjoint  
fa(m) man-loaded(m) intersect man-unloaded(m) = {}

assert man-mvrs-in-head-are-not-pre-ordered-1  
fa(m, mvr1, mvr2)  
mvr1 in man-head(m) & mvr2 in man-head(m)  
=> man-mvr-less?(m, mvr1, mvr2) = false

assert man-mvrs-in-head-are-not-pre-ordered-2  
fa(m, mvr1, mvr2)

```

    mvr1 in man-head(m) & mvr2 in man-head(m)
    => man-mvr-less?(m, mvr2, mvr1) = false

assert man-mvrs-in-head-preorder-mvrs-in-tail
fa(m, mvr1, mvr2)
mvr1 in man-head(m) & mvr2 in man-tail(m)
=> man-mvr-less?(m, mvr1, mvr2)

assert man-mvrs-in-tail-do-not-preorder-mvrs-in-head
fa(m, mvr1, mvr2)
    mvr1 in man-head(m) & mvr2 in man-tail(m)
    => man-mvr-less?(m, mvr2, mvr1) = false

assert man-mark-loaded-refines-manifest
fa(m, mvr)
    mvr in man-unloaded(m)
    => man-refined?(m, man-mark-mvr-loaded(m, mvr))

assert man-refinement-preserves-loaded-status
fa(m1, m2, mvr)
    man-refined?(man-mark-mvr-loaded(m1, mvr), m2)
    => man-mvr-has-been-loaded?(m2, mvr)

assert man-mark-loaded-implies-has-been-loaded
fa(m, mvr)
    mvr in man-unloaded(m)
    => man-mvr-has-been-loaded?(man-mark-mvr-loaded(m, mvr),
                                mvr)

END-THEORY

```

## A.5 Geometry

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)
% Represents the geometry of a yard.

THEORY GEOMETRY

% -----

THEORY-IMPORTS {SLOT, TIME-AND-DISTANCE}

```

% -----

## THEORY-TYPES

```
type geo = tuple(distances: seq(seq(distance)),
  count: fixnum)
```

% -----

## THEORY-OPERATIONS

```
function geo-make-empty(): geo
  = <[], 0>
```

```
function geo-next-number(g: geo): integer
  = g.count + 1
```

```
function geo-add-slot(g: geo, name: slot-name): tuple(geo, slot)
  = let(new-dists: seq(seq(distance))
    = append(image(lambda(s) append(s, *undefined*), g.distances),
      [ *undefined* | (i) i in [1 .. geo-next-number(g)]]))
  <<new-dists, geo-next-number(g)>,
    sl-make(geo-next-number(g), name)>
```

```
function geo-add-assymetric-separation(g: geo,
  src: slot-id, des: slot-id, dist: distance
): geo
  = let(new-dist: seq(seq(distance)) =
    seq-shadow1(g.distances, src,
      seq-shadow1(g.distances(src), des, dist)))
  <new-dist, g.count>
```

```
function geo-add-separation(g: geo,
  src: slot-id, des: slot-id, dist: distance
): geo
  = geo-add-assymetric-separation(
    geo-add-assymetric-separation(g, src, des, dist),
    des, src, dist)
```

```
function geo-slot-ids(g: geo): set(slot-id)
  = {1 .. g.count}
```

```
function geo-distance(g: geo, src: slot-id, des: slot-id
  | src in geo-slot-ids(g)
  & des in geo-slot-ids(g)
): distance
  = g.distances(src)(des)
```

END-THEORY

## A.6 Hostler

```
!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Represents a hostler.

THEORY HOSTLER

% -----

THEORY-IMPORTS {TIME-AND-DISTANCE, SLOT, CARGO,
                GEOMETRY, MOVEMENT-REQUEST}

% -----
THEORY-TYPES

type hostler-id = fixnum
type hostler-name = symbol
type hostler = tuple(id: hostler-id, name: hostler-name)

% -----

THEORY-OPERATIONS
function hos-make(id: hostler-id, name: hostler-name): hostler
  = <id, name>

function hos-id(h: hostler): hostler-id
  = h.id

function hos-speed(h: hostler, c: cargo): speed
  = *LOADED-HOSTLER-SPEED*

function hos-unloaded-speed(h: hostler): speed
  = *UNLOADED-HOSTLER-SPEED*

function hos-duration-for-move(h: hostler, mvr: mvr, g: geo): duration
  = add-durations(
      mvr-distance(mvr, g)
      div hos-speed(h, mvr-cargo(mvr)),
```

```

*TRIP-DURATION-OVERHEAD*)

function hos-duration-for-unloaded-trip(h: hostler,
    src: slot-id, des: slot-id, g: geo)
    : duration
    = geo-distance(g, src, des)
div hos-unloaded-speed(h)

function hos-duration-for-repositioning(h: hostler,
    mvr1: mvr, mvr2: mvr, g: geo)
    :duration
    = add-durations(
        hos-duration-for-unloaded-trip(h,
            mvr-destination(mvr1), mvr-source(mvr2), g),
        *REPOSITIONING-OVERHEAD*)

function hos-duration-for-move-with-repositioning(h: hostler,
    mvr: mvr, origin: slot-id, g: geo
    ): duration
    = add-durations(
        add-durations(
            hos-duration-for-move(h, mvr, g),
            hos-duration-for-unloaded-trip(h, origin, mvr-source(mvr), g)),
        *REPOSITIONING-OVERHEAD*)

END-THEORY

```

## A.7 Yard

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Represents a cargo yard: geometry, resources.

THEORY YARD

% -----

THEORY-IMPORTS {SLOT, HOSTLER, GEOMETRY}

% -----

THEORY-TYPES

```



```

type yard = tuple(
  slots: map(slot-id, slot),
  hostlers: map(hostler-id, hostler),
  geometry: geo,
  cached-hostlers: set(hostler),
  number-of-hostlers: fixnum
)

```

```

% -----

```

## THEORY-OPERATIONS

```

function y-make(S: set(slot), H: set(hostler), G: geo): yard
  = <{ | sl-id(sl) -> sl | (sl: slot) sl in S | },
    { | hos-id(ho) -> ho | (ho: hostler) ho in H | },
    G,
    H,
    size(H)
  >

```

```

function y-slot-ids(y: yard): set(slot-id)
  = domain(y.slots)

```

```

function y-slot(y: yard, s-id: slot-id
  | s-id in y-slot-ids(y)
): slot
  = y.slots(s-id)

```

```

function y-slot-from-name(y: yard, name: slot-name): slot
  = arb(filter(lambda(s: slot) sl-name(s) = name, range(y.slots)))

```

```

function y-hostler-ids(y: yard): set(hostler-id)
  = domain(y.hostlers)

```

```

function y-hostler(y: yard, h-id: hostler-id
  | h-id in y-hostler-ids(y)
): hostler
  = y.hostlers(h-id)

```

```

function y-hostlers(y: yard): set(hostler)
  = y.cached-hostlers

```

```

function y-geometry(y: yard): geo
  = y.geometry

```

```

function y-number-of-hostlers(Y: yard): fixnum
  = y.number-of-hostlers

```

END-THEORY

## A.8 Reservation

```
!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Represents a reservation of a hostler for a cargo item
% in a schedule.

THEORY RESERVATION

% -----

THEORY-IMPORTS {HOSTLER, MOVEMENT-REQUEST, TIME-AND-DISTANCE,
YARD}

% -----

THEORY-TYPES
type rvn = tuple(hostler-id: hostler-id,
                 mvr: mvr,
                 period: period)

% -----

THEORY-OPERATIONS
function rvn-make(h: hostler-id, mvr: mvr, p: period, Y: yard): rvn
  = <h, mvr, p>

function rvn-hostler-id(r: rvn): hostler-id
  = r.hostler-id

function rvn-mvr(r: rvn): mvr
  = r.mvr

function rvn-period(r: rvn): period
  = r.period

END-THEORY
```

## A.9 Schedule

```
!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

THEORY SCHEDULE

% -----

THEORY-IMPORTS {TIME-AND-DISTANCE, HOSTLER, MOVEMENT-REQUEST,
                RESERVATION, GEOMETRY, YARD, SHIP-MANIFEST}

% -----

THEORY-TYPE-PARAMETERS {}

% -----

THEORY-TYPES
type h-info = tuple(rvns: seq(rvn),
                    load: duration,
                    closure-time: time,
                    closure-slot: slot-id)

type schedule = tuple(hostlers: seq(h-info),
                      cached-h-ids: set(hostler-id),
                      closure: rvn,
                      sum-of-closure-times: time)

% -----

THEORY-OPERATIONS
function sch-make-empty(n: fixnum): schedule
= <[<[], 0, 0, undefined> | (i) i in [1 .. n]],
  {},
  undefined,
  0
>

function sch-is-empty?(S: schedule): boolean
= not(defined?(S.closure))

function sch-hostler-ids(S: schedule): set(hostler-id)
= S.cached-h-ids
```

```

function sch-size(S: schedule): fixnum
  = reduce(+,
    image(lambda(h-id) size(S.hostlers(h-id).rvns),
      sch-hostler-ids(S)),
    0)

function sch-sort-reservations(rs: set(rvn)): seq(rvn)
  = sort(rs, lambda(r1: rvn, r2: rvn) is-earlier-time?(
    pd-start(rvn-period(r1)),
    pd-start(rvn-period(r2))))

function sch-reservations-for-hostler(S: schedule, h-id: hostler-id)
  : set(rvn)
  = seq-to-set(S.hostlers(h-id).rvns)

function sch-ordered-reservations-for-hostler(S: schedule,
                                              h-id: hostler-id)
  : seq(rvn)
  = S.hostlers(h-id).rvns

function find-preceeding-reservation-aux(
  rvns: seq(rvn), ti: time, l: fixnum, u: fixnum
): rvn
  = if l=u
    then if l <= size(rvns) then rvns(l) else undefined
    else
      let(m: fixnum = (l+u) div 2)
      if is-earlier-time?(pd-end(rvn-period(rvns(m))), ti)
      then find-preceeding-reservation-aux(rvns, ti, m, u)
      else find-preceeding-reservation-aux(rvns, ti, l, m)

function find-preceeding-reservation(rvns: seq(rvn), ti: time): rvn
  = find-preceeding-reservation-aux(rvns, ti, 1, size(rvns))

function sch-all-reservations(S: schedule): set(rvn)
  = reduce(union, [ sch-reservations-for-hostler(S, h-id)
    | (h-id: hostler-id) h-id in set-to-seq(sch-hostler-ids(S))])

function sch-add-reservation(S: schedule, r: rvn): schedule
  = let(h-id: hostler-id = rvn-hostler-id(r))
    let(new-rvns: seq(rvn) = prepend(S.hostlers(h-id).rvns, r),
    new-load: duration = S.hostlers(h-id).load + pd-duration(rvn-period(r)),
    new-closure-time: time = pd-end(rvn-period(r)),
    new-closure-slot: slot-id = mvr-destination(rvn-mvr(r)))
    let(new-closure: rvn =

```

```

    if not(defined?(S.closure))
    then r
    else if new-closure-time > pd-end(rvn-period(S.closure))
        then r
        else S.closure,
new-sum-of-closure-times: time
  = S.sum-of-closure-times
    + new-closure-time
    - S.hostlers(h-id).closure-time)
  <seq-shadow1(S.hostlers, h-id,
<new-rvns, new-load, new-closure-time, new-closure-slot > ),
  S.cached-h-ids with h-id,
  new-closure,
  new-sum-of-closure-times >

function sch-sum-of-closure-times(S: schedule): time
  = S.sum-of-closure-times

function sch-closure-slot-for-hostler(S: schedule, h-id: hostler-id)
  : slot-id
  = S.hostlers(h-id).closure-slot

function sch-extract-mvrs-from-seq-of-reservations(rs: seq(rvn))
  : seq(mvr)
  = image(lambda(r:rvn) rvn-mvr(r), rs)

function sch-last-reservation-for-hostler(S: schedule, h-id: hostler-id)
  : rvn
  = first(S.hostlers(h-id).rvns)

function sch-closure-time-for-hostler(S: schedule, h-id: hostler-id)
  : time
  = S.hostlers(h-id).closure-time

function sch-last-reservations(S: schedule): seq(rvn)
  = [ sch-last-reservation-for-hostler(S, h-id)
    | (h-id: hostler-id) h-id in set-to-seq(sch-hostler-ids(S)) ]

function sch-closure-reservation(S: schedule): rvn
  = S.closure

function sch-closure-time(S: schedule): time
  = let(r: rvn = sch-closure-reservation(S))
    if defined?(r)
    then pd-end(rvn-period(r))
    else 0

```

```

function sch-rvns-are-successive(S: schedule, r1: rvn, r2: rvn)
  : boolean
= let(h-id: hostler-id = rvn-hostler-id(r1))
  h-id = rvn-hostler-id(r2)
  and is-earlier-time?(pd-start(rvn-period(r1)),
pd-start(rvn-period(r2)))
  and not(ex(r3: rvn)(
    r3 in sch-reservations-for-hostler(S, h-id)
    and is-earlier-time?(pd-start(rvn-period(r1)),
pd-start(rvn-period(r3)))
    and is-earlier-time?(pd-start(rvn-period(r3)),
pd-start(rvn-period(r2)))
  ))

```

```

function sch-duration-for-hostler(S: schedule, h-id: hostler-id)
  : duration
= S.hostlers(h-id).load

```

```

function sch-total-duration(S: schedule): duration
= reduce(add-durations,
  [ sch-duration-for-hostler(S, h-id)
    | (h-id) h-id in set-to-seq(sch-hostler-ids(S))],
0)

```

END-THEORY

## A.10 Work Metric

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

```

```

% Metrics on schedules, based on "work done"
% (i.e., cargo shipped, distance moved).
% Used for pruning schedules.

```

THEORY METRIC-WORK

```

% -----
THEORY-IMPORTS {HOSTLER, MOVEMENT-REQUEST, RESERVATION,
GEOMETRY, YARD, SHIP-MANIFEST, SCHEDULE, TIME-AND-DISTANCE}
% -----

```

## THEORY-TYPES

type hsec = fixnum

% -----

## THEORY-OPERATIONS

function load-based-on-hostler(man: manifest, h: hostler, g: geo): hsec  
= man-unloaded-distance(man)

function true-repositioning-cost-for-hostler(  
S: schedule, h-id: hostler-id, Y: yard  
) : hsec  
= let(ordered: seq(rvn) = sch-ordered-reservations-for-hostler(S, h-id))  
let(h: hostler = y-hostler(Y, h-id))  
reduce(+,  
[ hos-duration-for-repositioning(h,  
rvn-mvr(ordered(i - 1)), rvn-mvr(ordered(i)), y-geometry(Y))  
| (i: integer) i in [2 .. size(ordered)]] ,  
0)

function true-total-repositioning-cost(S: schedule, Y: yard): hsec  
= reduce(+,  
[ true-repositioning-cost-for-hostler(S, h-id, Y)  
| (h-id: hostler-id) h-id in set-to-seq(sch-hostler-ids(S)) ],  
0)

function load-cost-for-hostler(S: schedule, h-id: hostler-id): hsec  
= sch-duration-for-hostler(S, h-id)

function total-load-cost(S: schedule): hsec  
= sch-total-duration(S)

function work-cost-for-hostler(S: schedule, h-id: hostler-id): hsec  
= sch-closure-time-for-hostler(S, h-id)

function total-work-cost(S: schedule): hsec  
= sch-sum-of-closure-times(S)

function total-work-cost-precompute(S: schedule, works: seq(hsec))  
: hsec  
= reduce(+,  
[ works(h-id)  
| (h-id) h-id in set-to-seq(sch-hostler-ids(S))])

function work\*(S: schedule): hsec

```

= let(r: rvn = sch-closure-reservation(S))
  if defined?(r)
  then pd-end(rvn-period(r))
  else 0

function work*-precompute(S: schedule, works: seq(hsec)): hsec
= reduce(lambda(x: hsec, y: hsec)
  if x >= y then x else y,
  [ works(h-id)
    | (h-id: hostler-id) h-id in set-to-seq(sch-hostler-ids(S)) ],
  0)

function work*-id(S: schedule): hostler-id
= arb({h-id | (h-id: hostler-id) h-id in sch-hostler-ids(S)
  and work-cost-for-hostler(S, h-id) = work*(S)})

function repositioning-cost-for-hostler(S: schedule, h-id: hostler-id)
: hsec
= work-cost-for-hostler(S, h-id) - load-cost-for-hostler(S, h-id)

function total-repositioning-cost(S: schedule): hsec
= total-work-cost(S) - total-load-cost(S)

function schedule-cost(S: schedule, Y: yard): hsec
= work*(S) * y-number-of-hostlers(Y)

function schedule-cost-precompute(S: schedule, Y: yard,
  works: seq(hsec)): hsec
= work*-precompute(S, works) * size(y-hostler-ids(y))

function spare(S: schedule, Y: yard): hsec
= schedule-cost(S, Y) - total-work-cost(S)

function find-shortest-distances(
  d: slot-id, c: integer, ss: seq(slot-id), g: geo
): seq(distance)
= let(sorted: seq(distance)
= sort([geo-distance(g, d, s) | (s) s in ss],
  lambda(x,y) x <= y))
  [sorted(i) | (i) i in [1 .. c]]

function find-shortest-distances-for-source(
  s: slot-id, c: integer, dd: seq(slot-id), g: geo
): seq(distance)
= let(sorted: seq(distance)
= sort([geo-distance(g, d, s) | (d) d in dd],

```



```

    lambda(x,y) x <= y))
[sorted(i) | (i) i in [1 .. c]]

function repo-lower-bound-from-destinations(S: schedule,
                                           Y: yard, M: manifest)
: hsec
= man-repositioning-lower-bound(M, Y) div *UNLOADED-HOSTLER-SPEED*
  + *REPOSITIONING-OVERHEAD*
  * (size(man-unloaded(M)) - y-number-of-hostlers(Y))

function repo-lower-bound(S: schedule, Y: yard, M: manifest): hsec
= let(d2 = repo-lower-bound-from-destinations(S, Y, M))
  d2

function find-largest-distances(
  d: slot-id, c: integer, ss: seq(slot-id), g: geo
): seq(distance)
= let(sorted: seq(distance)
= sort([geo-distance(g, d, s) | (s) s in ss],
      lambda(x,y) x >= y))
[sorted(i) | (i) i in [1 .. c]]

function predicted-cost-pre-compute-repo(
  S: schedule, Y: yard, M: manifest, repo: distance, load: hsec
): hsec
= if man-finished?(M)
  then schedule-cost(S, Y)
  else
    let(w: hsec = total-work-cost(S),
cost: hsec = schedule-cost(S, Y))
    let(spare-cost: hsec = cost - w)
    if load + repo > spare-cost
    then cost + load + repo - spare-cost
    else cost

function predicted-cost(S: schedule, Y: yard, M: manifest): hsec
= predicted-cost-pre-compute-repo(S, Y, M, repo-lower-bound(S, Y, M),
  load-based-on-hostler(M,
  arb(y-hostlers(Y)), y-geometry(Y)))

function lower-bound-for-delta-schedule(
  old-S: schedule, h-id: hostler-id, new-M: manifest,
  Y: yard, repo: distance, load: hsec,
  rvn-h: rvn
): hsec
= let(fts-h: time = pd-end(rvn-period(rvn-h)))

```

```

    let(new-closure-time: time
= greater-i(sch-closure-time(old-S), fts-h))
    let(new-cost-S: hsec = new-closure-time * y-number-of-hostlers(Y))
    if man-finished?(new-M)
    then new-cost-S
    else
        let(w: hsec
= sch-sum-of-closure-times(old-S)
+ fts-h - sch-closure-time-for-hostler(old-S, h-id))
        let(spare-cost: hsec = new-cost-S - w)
        if load+repo > spare-cost
        then load + repo + w
        else new-cost-S

```

END-THEORY

## A.11 YMS

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

```

% Defines the constraints on a feasible hostler schedule.

THEORY YARD-MANAGEMENT-SYSTEM-2

% -----

THEORY-IMPORTS {HOSTLER, MOVEMENT-REQUEST, RESERVATION,  
GEOMETRY, YARD, SHIP-MANIFEST, SCHEDULE}

% -----

THEORY-TYPE-PARAMETERS {}

% -----

THEORY-TYPES

% -----

THEORY-OPERATIONS

```

function sch-sequence-of-mvr-refined?(s1: seq(mvr), s2: seq(mvr))
: boolean

```

```

= size(s1) <= size(s2)
  and-then fa(i: fixnum)( i in domain(s1) => s1(i) = s2(i) )

function schedule-refined?(S1: schedule, S2: schedule): boolean
= sch-hostler-ids(S1) subset sch-hostler-ids(S2)
  and-then fa(h-id: hostler-id)
    (h-id in sch-hostler-ids(S1)
      => sch-sequence-of-mvr-refined?(
        sch-extract-mvrs-from-seq-of-reservations(
          sch-ordered-reservations-for-hostler(S1, h-id)),
        sch-extract-mvrs-from-seq-of-reservations(
          sch-ordered-reservations-for-hostler(S2, h-id)))
    )

function order-respected?(S: schedule, man: manifest): boolean
= fa(r1: rvn, r2: rvn)
  (r1 in sch-all-reservations(S)
    and r2 in sch-all-reservations(S)
      => man-mvr-less?(man, rvn-mvr(r1), rvn-mvr(r2))
        => is-earlier-time?(pd-end(rvn-period(r1)),
          pd-end(rvn-period(r2)))
    )

function feasible-resources?(S: schedule, Y: yard): boolean
= sch-hostler-ids(S) subset y-hostler-ids(Y)

function feasible-durations?(S: schedule, Y: yard): boolean
= fa(r: rvn)(r in sch-all-reservations(S)
  => is-shorter-duration?(
    hos-duration-for-move(
      y-hostler(Y, rvn-hostler-id(r)),
      rvn-mvr(r),
      y-geometry(Y)),
    pd-duration(rvn-period(r)))
  )

function feasible-separations?(S: schedule, Y: yard): boolean
= fa(r1: rvn, r2: rvn)(
  r1 in sch-all-reservations(S)
    and r2 in sch-all-reservations(S)
      and sch-rvns-are-successive(S, r1, r2)
        => is-later-time?(
          pd-start(rvn-period(r2)),
          add-duration-to-time(
            pd-end(rvn-period(r1)),
            hos-duration-for-repositioning(

```

```

        y-hostler(Y, rvn-hostler-id(r1)),
        rvn-mvr(r1),
        rvn-mvr(r2),
        y-geometry(Y))))
    )

function YMS-Schedule(Y: yard, man: manifest | true)
returns (S: schedule
    | order-respected?(S, man)
    & feasible-durations?(S, Y)
    & feasible-separations?(S, Y)
    & feasible-resources?(S, Y)
    )

END-THEORY

```

## A.12 Global Search with Pruning

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Theory for global search with pruning.

THEORY SCHED-GS-PRUNING

% -----

THEORY-IMPORTS {HOSTLER, MOVEMENT-REQUEST, RESERVATION,
GEOMETRY, YARD, SHIP-MANIFEST, SCHEDULE,
YARD-MANAGEMENT-SYSTEM-2, SCHED-TESTER,
SCHED-GREEDY-APPEND-FOR-EARLIEST-MVR-FINISH}

% -----

THEORY-TYPE-PARAMETERS {}

% -----

THEORY-TYPES
var *tree-size*: fixnum = 0
var *leaf-size*: fixnum = 0
var *lb*: hsec = 999999
var *opt*:real = 2.0

```

```

var answer: schedule = undefined

% -----

THEORY-OPERATIONS
function fts(S: schedule, Y: yard, h-id: hostler-id, mvr: mvr,
  lower-bound: time, mvr-duration: duration)
  : time
  = let(close:time =
if h-id in sch-hostler-ids(S)
then sch-closure-time-for-hostler(S, h-id)
  + mvr-duration
  + geo-distance(y-geometry(Y),
sch-closure-slot-for-hostler(S, h-id),
mvr-source(mvr))
  div *UNLOADED-HOSTLER-SPEED*
  + *REPOSITIONING-OVERHEAD*
else add-duration-to-time(*EARLIEST-TIME*, mvr-duration))
  if is-later-time?(lower-bound, close)
  then lower-bound
  else close

function find-latest-reservation-of-lower-rank-aux(
  rvns: seq(rvn), man: manifest, m: mvr, l: fixnum
): rvn
  = if l > size(rvns) then undefined
  elseif man-mvr-less?(man, rvn-mvr(rvns(l)), m)
  then rvns(l)
  else find-latest-reservation-of-lower-rank-aux(rvns, man,
                                                    m, l+1)

function find-latest-reservation-of-lower-rank(
  rvns: seq(rvn), man: manifest, m: mvr
): rvn
  = find-latest-reservation-of-lower-rank-aux(rvns, man, m, 1)

function find-latest-reservation-of-lower-rank-for-hostler(
  S: schedule, h-id: hostler-id, man: manifest, m: mvr
): rvn
  = let(rvns: seq(rvn)
    = sch-ordered-reservations-for-hostler(S, h-id))
    find-latest-reservation-of-lower-rank(rvns, man, m)

function find-latest-reservations-of-lower-rank(
  S: schedule, man: manifest, m: mvr
): seq(rvn)

```

```

= [ r | (r: rvn, h-id: hostler-id)
      h-id in set-to-seq(sch-hostler-ids(S))
& r = find-latest-reservation-of-lower-rank-for-hostler(S, h-id, man, m)
& defined?(r) ]

```

```

function make-new-schedule-for-hostler(
  S: schedule, mvr: mvr, h-id: hostler-id, Y: yard, M: manifest,
  earliest-end: time, mvr-duration: duration
): schedule
= let(fts-h = fts(S, Y, h-id, mvr, earliest-end, mvr-duration))
  let(rvn-h = rvn-make(h-id,
mvr,
pd-make-from-times(
  fts-h - mvr-duration,
  fts-h),
Y))
  let(new-S = sch-add-reservation(S, rvn-h))
  new-S

```

```

function gs-aux(S: schedule, M: manifest, Y: yard,
  opt-factor: real, depth: fixnum)
: schedule
= *tree-size* <- *tree-size* + 1;
  if man-finished?(M)
  then (*leaf-size* <- *leaf-size* + 1;
if schedule-cost(S, Y) < *lb*
then (*lb* <- schedule-cost(S, Y);
    print([*tree-size*, *lb*]);
    S)
else undefined
)
  else
  let(mvr-choice: mvr = arb(man-head(M)))
  let(manifests: manifest
= man-mark-mvr-loaded(M, mvr-choice))
  let(repos: hsec = repo-lower-bound(S, Y, manifests))
  let(pre-mvrs: seq(rvn)
= find-latest-reservations-of-lower-rank(S, M, mvr-choice))
  let(earliest-end: time =
max-seq-i(append([pd-end(rvn-period(pre-mvrs(i)))
  | (i) i in domain(pre-mvrs)],
*EARLIEST-TIME*)))
  let(mvr-duration: duration
= mvr-distance(mvr-choice, y-geometry(Y)) div *LOADED-HOSTLER-SPEED*
+ *TRIP-DURATION-OVERHEAD*)
  let(loads: hsec

```

```

= load-based-on-hostler(manifests, arb(y-hostlers(Y)), y-geometry(Y))
  let(rvns: map(hostler-id, rvn)
= { | h-id -> r | (h-id: hostler-id, r: rvn, fts-h: time)
  h-id in set-to-seq(y-hostler-ids(Y))
  & fts-h = fts(S, Y, h-id, mvr-choice, earliest-end, mvr-duration)
  & r = rvn-make(h-id, mvr-choice,
  pd-make-from-times(fts-h - mvr-duration, fts-h),
  Y) | })
  let(lower-bounds: seq(tuple(mvr, hostler-id, hsec)) =
sort([ < mvr-choice, h-id, lower-bound >
  | (h-id: hostler-id, lower-bound: hsec,
new-M: manifest)
  h-id in set-to-seq(y-hostler-ids(Y))
  & h-id <= depth
  & new-M = manifests
  & lower-bound =
    lower-bound-for-delta-schedule(
S, h-id, new-M, Y, repos, loads, rvns(h-id))
  ],
  lambda(smc1: tuple(mvr, hostler-id, hsec),
smc2: tuple(mvr, hostler-id, hsec))
smc1.3 <= smc2.3))
let(schedules: seq(tuple(schedule, hsec)) =
[ < new-S, cost >
  | (smc: tuple(mvr, hostler-id, hsec),
  h-id: hostler-id, lower-bound: hsec,
  new-S: schedule, cost: hsec)
  smc in lower-bounds
  & h-id = smc.2 & lower-bound = smc.3
  & new-S = (if lower-bound < *lb* * opt-factor
  then gs-aux(sch-add-reservation(S, rvns(h-id)),
manifests, Y, opt-factor, depth+1)
  else (*tree-size* <- *tree-size* + 1;
  undefined))
  & defined?(new-S)
  & cost = if defined?(new-S)
    then schedule-cost(new-S, Y)
  else undefined
  ])
  let(optimum: tuple(schedule, hsec) =
if not(empty(schedules))
then reduce(lambda(sm1: tuple(schedule, hsec),
sm2: tuple(schedule, hsec))
  if sm1.2 <= sm2.2
then sm1 else sm2,
schedules)

```

```

else <undefined, undefined>)
    optimum.1

function gs(M: manifest, Y: yard, opt: real): schedule
    = gs-aux(sch-make-empty(size(y-hostlers(Y))), M, Y,
        1.0/(1.0 + opt/100.0), 1)

function test-schedule(opts: seq(real))
    = let(Y = construct-yard())
        let(M = construct-manifest(Y))
        let(initial: schedule
            = greedy-append-for-earliest-mvr-finish(M, Y))
        let(lb = schedule-cost(initial, Y))
        image(lambda(opt: real)
            (*lb* <- lb;
            *tree-size* <- 0;
            *leaf-size* <- 0;
            answer <- gs(M, Y, opt);
            (if defined?(answer)
                then print([
lb,
real-to-nearest-integer(opt), *tree-size*, *leaf-size*,
schedule-cost(answer, Y),
total-load-cost(answer),
true-total-repositioning-cost(answer, Y),
total-repositioning-cost(answer)
- true-total-repositioning-cost(answer, Y),
spare(answer, Y)]))
            else print([lb, real-to-nearest-integer(opt),
                *tree-size*, *leaf-size*, undefined]));
        print("");
        undefined),
        opts)

```

END-THEORY

## A.13 Greedy Heuristic Scheduler

```

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'cypress-grammar)

% Theory for greedy heuristic scheduler
% based on earliest finishing time.

```



## THEORY SCHED-GREEDY-APPEND-FOR-EARLIEST-MVR-FINISH

% -----

THEORY-IMPORTS {TIME-AND-DISTANCE, CARGO, SLOT, GEOMETRY,  
MOVEMENT-REQUEST, HOSTLER, SHIP-MANIFEST, YARD,  
RESERVATION, SCHEDULE, YARD-MANAGEMENT-SYSTEM-2, SCHED-  
TESTER, METRIC-WORK}

% -----

### THEORY-OPERATIONS

function separation-time(Y: yard, h-id: hostler-id,  
mvr1: mvr, mvr2: mvr)  
: duration  
= hos-duration-for-repositioning(y-hostler(Y, h-id),  
mvr1, mvr2, y-geometry(Y))

function duration-for-mvr(Y: yard, h-id: hostler-id, mvr: mvr)  
: duration  
= hos-duration-for-move(y-hostler(Y, h-id), mvr, y-geometry(Y))

function duration-for-move-with-repositioning(Y: yard, h-id: hostler-id,  
mvr1: mvr, mvr2: mvr): duration  
= hos-duration-for-move-with-repositioning(y-hostler(Y, h-id),  
mvr2, mvr-destination(mvr1), y-geometry(Y))

function fts1(S: schedule, Y: yard, h-id: hostler-id, mvr: mvr,  
lower-bound: time)  
: time  
= let(close:time =  
if h-id in sch-hostler-ids(S)  
then add-duration-to-time(  
sch-closure-time-for-hostler(S, h-id),  
duration-for-move-with-repositioning(  
Y, h-id,  
rvn-mvr(sch-last-reservation-for-hostler(S, h-id)),  
mvr))  
else add-duration-to-time(  
\*EARLIEST-TIME\*,  
duration-for-mvr(Y, h-id, mvr))  
)  
if is-later-time?(lower-bound, close) then lower-bound else close

function greedy-append-for-earliest-mvr-finish-aux(

```

S: schedule, M: manifest, Y: yard
): schedule
= if man-finished?(M)
  then S
  else
    let(mvr*: mvr = arb(man-head(M)))
    let(pre: set(rvn))
= seq-to-set(find-latest-reservations-of-lower-rank(S, M, mvr*))
  let(earliest-end: time
= max-i({pd-end(rvn-period(rvn)) | (rvn: rvn) rvn in pre}
  with *EARLIEST-TIME*))
  let(fts*: time = min-i({fts1(S, Y, h-id, mvr*, earliest-end)
    | (h-id: hostler-id) h-id in y-hostler-ids(Y)}))
  let(h-id*: hostler-id
    = arb({h-id | (h-id: hostler-id) h-id in y-hostler-ids(Y)
      and fts1(S, Y, h-id, mvr*, earliest-end) = fts*}))
  let(rvn*: rvn = rvn-make(h-id*,
mvr*,
pd-make-from-times(
  subtract-duration-from-time(fts*,
    duration-for-mvr(Y, h-id*, mvr*)),
  fts*),
Y))
  let(new-S: schedule = sch-add-reservation(S, rvn*),
new-M: manifest = man-mark-mvr-loaded(M, mvr*))
  greedy-append-for-earliest-mvr-finish-aux(new-S, new-M, Y)
function greedy-append-for-earliest-mvr-finish(M: manifest, Y: yard)
  : schedule
= let (empty-sch = sch-make-empty(size(y-hostlers(Y))))
  greedy-append-for-earliest-mvr-finish-aux(empty-sch, M, Y)

```

END-THEORY